Solucionando Sudoku com Simulated Annealing

Ricardo Godoy de Oliveira
rgoliveira@inf.ufrgs.br

Junho de 2013

Conteúdo

1 Introdução

Sudoku — pronunciado Sudôku, em português — é uma espécie de quebra-cabeça onde o objetivo é preencher uma matriz $n^2 \times n^2$ de tal modo que na mesma linha, coluna e submatriz $n \times n$ o mesmo número não se repita. Foi projetado por Howard Garns, acredita-se que baseado no quadrado latino, trabalho de Leonhard Euler.
No quadrado latino, temos uma matriz $n \times n$ preenchida com n simbolos, cada um ocorrendo exatamente uma vez em cada linha e cada coluna. Sabe-se que o problema de completar um quadrado latino parcialmente preenchido é NP-Completo, e portanto o Sudoku também o é, já que as instâncias do último são um caso especial de quadrado latino.
As primeiras edições do Sudoku surgiram nos Estados Unidos na década de 1970. Anos depois, na década de 1980, uma importante empresa japonesa de quebra-cabeças descobriu o jogo e o popularizou no Japão, onde se tornou um dos jogos mais vendidos. Hoje é jogado por milhões de pessoas no mundo todo.
A forma mais frequente do Sudoku é a tabela de tamanho 9×9, embora também existam tabelas de tamanhos 16×16 e 25×25. Esta última é conhecida como *Samurai Sudoku*, devido à sua notável dificuldade em relação às outras menores. Além de tamanhos variados, também existem variações com regras diferentes; algumas variantes são apresentadas em [4].

2 Definição do problema

O problema que será discutido aqui será o de completar uma instância de Sudoku que já possua alguns valores definidos. Tais valores serão denominados *células* e serão fixos, ou seja, não poderão ser alterados em momento algum.
De maneira mais formal, podemos dar a seguinte definição ao problema:

Sudoku. *Dada uma matriz $n^2 \times n^2$, preenchê-la utilizando os números $i \in [1, n^2]$, de tal modo que:*

1. *Cada linha da matriz de células contém cada número $i \in [1, n^2]$ exatamente uma vez;*
2. *Cada coluna da matriz de células contém cada número $i \in [1, n^2]$ exatamente uma vez;*
3. *Cada submatriz $n \times n$ contém cada número $i \in [1, n^2]$ exatamente uma vez.*

3 Simulated Annealing

A metaheurística *Simulated Annealing* — ou Recozimento Simulado — tem base em uma analogia com um processo físico onde se busca um estado de baixa energia em um dado sólido: primeiramente, o material é aquecido até altas temperaturas, e então ele é resfriado tão lentamente quanto for necessário para que os átomos do sólido acabem se organizando de maneira uniforme, até um estado de energia mínima. Esse processo é utilizado para que o material final tenha uma energia interna baixa, o que diminui a possibilidade de defeitos [7, 6].

Para realizar o processo de arrefecimento, o ponto crucial é o controle da temperatura. É com isso que se atinge o estado desejado do sistema. Caso a temperatura caia rápido demais, talvez isso acarrete em defeitos no material ou em um estado que não é o de energia interna mínima. Mas caso a temperatura seja controlada de forma adequada, o sólido final irá adquirir uma estrura ótima, contendo um valor mínimo de energia interna.
A metaheurística traz a idéia da temperatura para a solução de problemas de otimização. Através dela tentamos escapar de mínimos locais por um mecanismo que, com uma probabilidade baseada nesta temperatura, permite a aceitação de uma solução inferior à atual; e conforme a temperatura é reduzida, esta probabilidade diminui. Deste modo, ao início do processo, estaremos aceitando praticamente qualquer solução do espaço de busca; ao final, estaremos aceitando apenas soluções que demonstrem melhoria.

3.1 Algoritmo

Na implementação da técnica de Simulated Annealing, o algoritmo de Metropolis [9, p. 160-3] [8] é utilizado para controlar a geração e a aceitação de novas soluções.

A partir de uma solução existente s e uma dada temperatura T, o algoritmo de Metropolis tem a seguinte forma:

Entrada: s, T
Saída: Solução s' com $c(s') \leq c(s)$
repita
  seleciona $s' \in N(s)$ aleatoriamente;
  seja $\Delta \leftarrow c(s') - c(s)$;
  se $\Delta \leq 0$ **então**
    atualiza $s \leftarrow s'$;
  senão
    atualiza $s \leftarrow s'$ com probabilidade $e^{-(\Delta/T)}$;
  fim
até *critério de parada é satisfeito*;
retorna s;

Algoritmo 1: Metropolis

Onde:

- $N(s)$ é a vizinhança de s;
- $c(s)$ é o custo da solução s.

Utilizando este algoritmo, podemos formular uma técnica para Simulated Annealing como segue:

Entrada: s, T, α, sl
Saída: Solução s' com $c(s') \leq c(s)$

```
repita
    para sl vezes faça
        seleciona s' ∈ N(s) aleatoriamente;
        seja Δ ← c(s') − c(s);
        se Δ ≤ 0 então
            atualiza s ← s';
        senão
            atualiza s ← s' com probabilidade e^{−(Δ/T)};
        fim
    fim
    T ← α.T
até critério de parada é satisfeito;
retorna s;
```

Algoritmo 2: Simulated Annealing

Onde:

- s é a solução atual;
- T é a temperatura inicial;
- α é a taxa de resfriamento;
- sl é o comprimento de cada estágio, ou seja, o número de iterações a serem executadas para cada nível de temperatura.

4 Aplicação ao problema

Para que seja possível aplicar a metaheurística, antes é necessário definir algumas coisas, como por exemplo a representação do problema, operador de vizinhança, entre outras. Estas definições serão explicadas a seguir.

4.1 Solução Inicial

Uma das entradas do Simulated Annealing é uma solução atual. Sendo assim, antes mesmo de iniciar o processo, se faz necessária uma solução inicial para suprir esta entrada.
Na implementação que fiz, esta solução inicial é gerada respeitando a terceira restrição[3] do jogo. Portanto, a idéia é que durante todo o processo esta restrição é garantida.

4.2 Vizinhança

O operador de vizinhança é necessário para gerar uma nova solução, e esta será ou não aceita conforme o algoritmo de Metropolis.
Para tal, como a solução inicial gerada respeita a terceira restrição[3], o operador de vizinhança deve manter esta propriedade. Portanto, o operador de vizinhança que defini escolhe, dentro de uma mesma submatriz $n \times n$, duas células **não-fixas** e as troca de lugar (*swap*).

4.3 Função objetivo

Levando em consideração as duas primeiras restrições[1][2], é evidente que, quando ambas são satisfeitas, não haverão números repetidos na mesma linha ou coluna. Sendo assim, dados

$$x_{zi} = \text{número de vezes que } z \in [1, n^2] \text{ se repete na linha } i \in [1, n^2]$$

e

$$y_{zj} = \text{número de vezes que } z \in [1, n^2] \text{ se repete na coluna } j \in [1, n^2]$$

a função objetivo é

$$Custo(s) = \sum_{z=1}^{n^2} \sum_{i=1}^{n^2} x_{zi} + \sum_{z=1}^{n^2} \sum_{j=1}^{n^2} y_{zj}$$

Isso irá funcionar bem pois, como a Solução Inicial e a Vizinhança sempre respeitam a terceira restrição[3], basta que a função objetivo avalie o quanto as linhas e colunas distam de uma solução ótima. Sendo assim, uma solução ótima irá possuir o valor zero.

5 Testes

5.1 Calibrando parâmetros

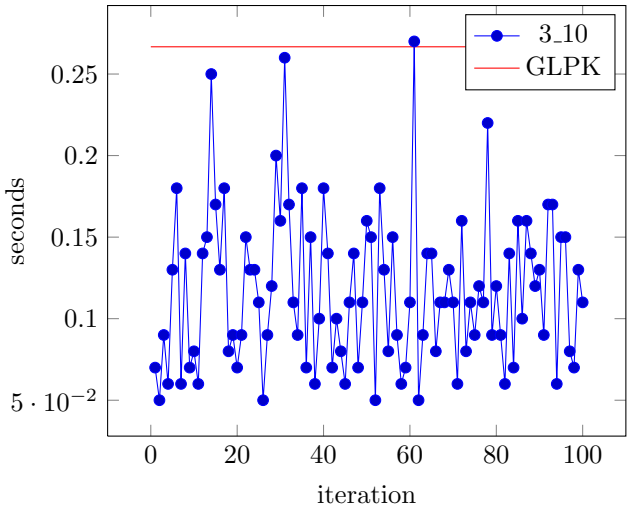
Para calibrar os parâmetros da metaheurítica foram realizados diversos testes aleatórios, além de uma pesquisa na literatura para tentar encontrar valores de referência ou até mesmo alguma forma sistemática de fazer a calibragem. Embora alguns autores tenham sugestões de calibragem inicial (vide [5, Seção 1.7 (pg 44,45)])), os valores são muito sensíveis ao problema e às instâncias.
Após diversos testes, foram encontrados os seguintes valores gerais para os parâmetros:

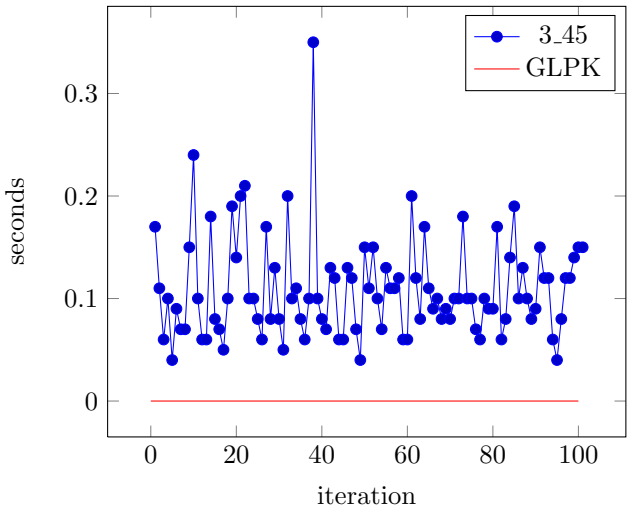
- Temperatura inicial: $n \times 100$
- α: 0.9 (utilizando a lei $T_{k+1} = \alpha \times T_k$)
- Iterações por estágio de temperatura: n^2

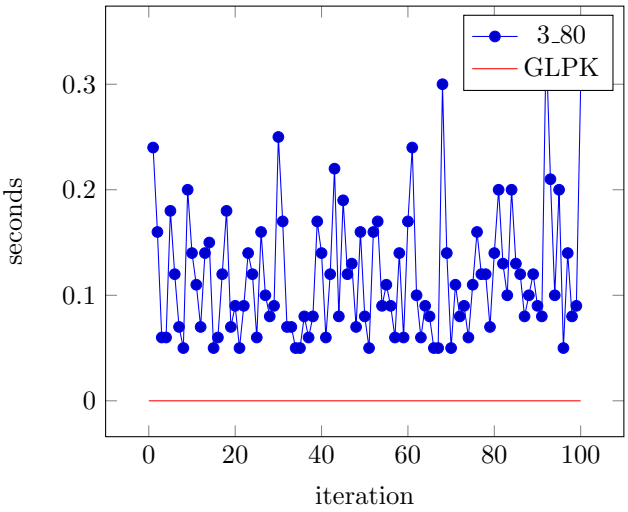
5.2 Resultados

A seguir estão dispostos os gráficos apresentando o tempo de execução da metaheurística ao longo de 100 testes para cada instância fornecida. Também está presente nos gráficos, para fins de comparação, uma reta representando a média do tempo de execução do solucionador genérico do GLPK.

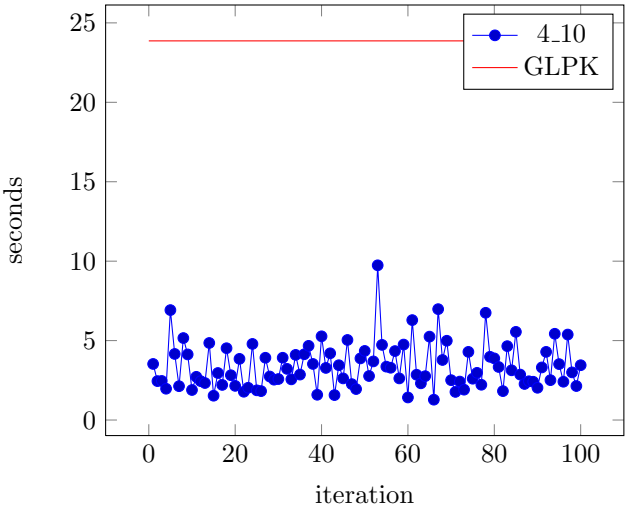
5.2.1 Instâncias de ordem 3

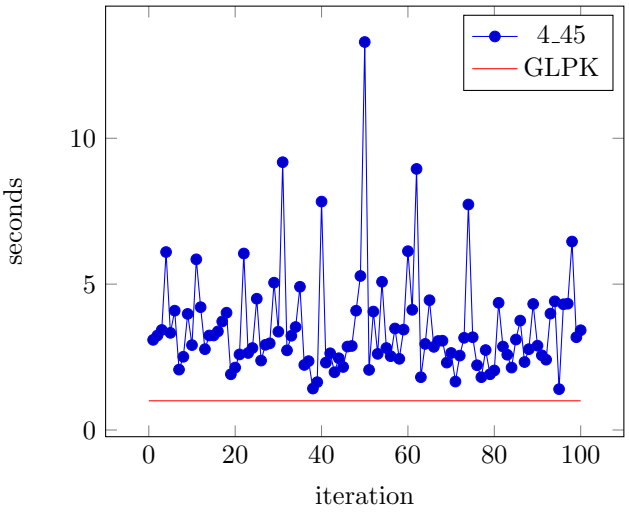


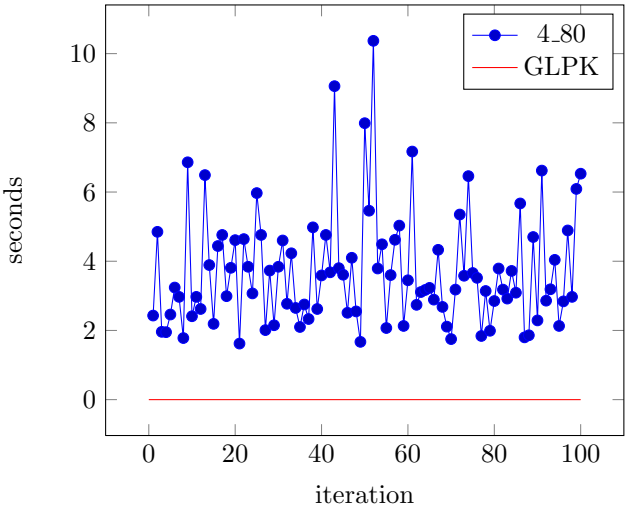




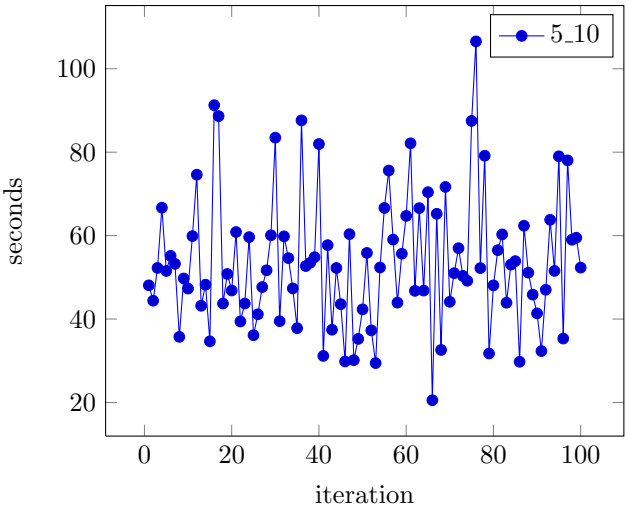
5.2.2 Instâncias de ordem 4



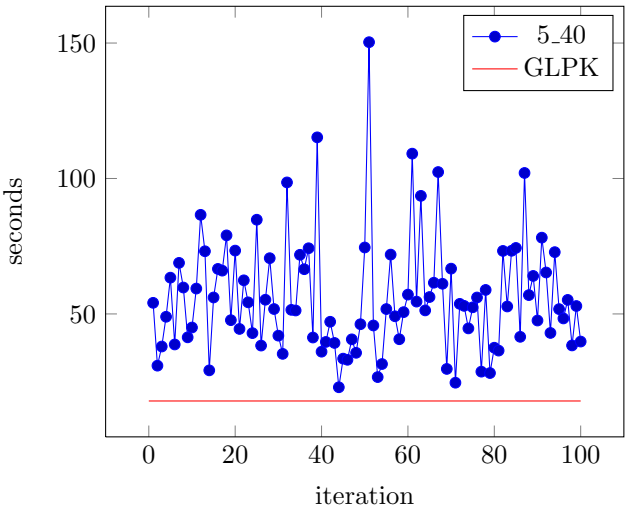

4_45
GLPK
seconds
iteration
0
5
10
0
20
40
60
80
100

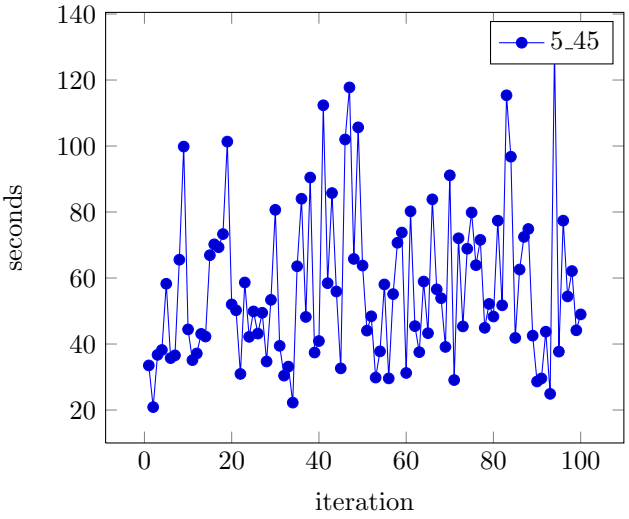

4_80
GLPK
seconds
iteration
0
2
4
6
8
10
0
20
40
60
80
100

5.2.3 Instâncias de ordem 5

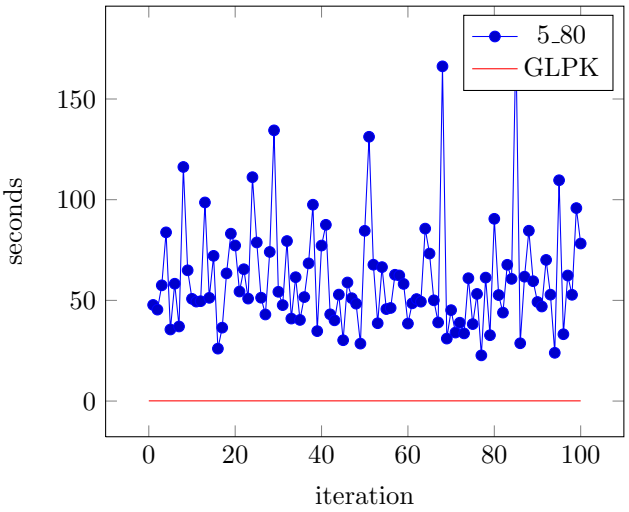


Neste caso acima o GLPK levou 773.5000 segundos para resolver. Não está no gráfico pois é muito distante da metaheurística.





Neste caso acima o GLPK levou 2058.8667 segundos para resolver. Não está no gráfico pois é muito distante da metaheurística.



6 Conclusões

A técnica de Simulated Annealing é bastante simples de se implementar e a convergência é garantida. Por esse ângulo, é uma excelente metaheurística e pode ser considerada para uma diversidade de problemas. Por outro lado, possui a grave desvantagem de ser difícil de calibrar, pois seus parâmetros são muito sensíveis, o que resulta em um tempo muito grande de calibragem para que se tenha confiança no valores encontrados. Ainda assim, a experiência de implementar esse solucionador foi bastante interessante e creio que agregou um conhecimento que me será muito útil em diversas outras situações.

Referências

[1] Marcos C. Machado, Luiz Chaimowicz. Computer Science Department, Federal University of Minas Gerais, Belo Horizonte, Brazil. *Combining Metaheuristics and CSP Algorithms to solve Sudoku.*

[2] Andrew C. Barlett, Amy N. Langville. Department of Mathematics, College of Charleston, Charlston, SC. *An Integer Programming Model for the Sudoku Problem.*

[3] Rhyd Lewis. Centre for Emergent Computing, Napier University, Scotland. *Metaheuristics can Solve Sudoku Puzzles.*

[4] Ed Pegg Jr, 2005/09/06. *Sudoku Variations.* Acesso em Junho de 2013 à http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html.

[5] J. Dréo, A. Pétrowski, E. Taillard. *Metaheuristics for Hard Optimization* (2006).

[6] Wikipedia. *Simulated Annealing.* Acesso em Junho de 2013 à http://en.wikipedia.org/wiki/Simulated_annealing.

[7] Wikipédia. *Simulated Annealing ou Arrefecimento simulado.* Acesso em Junho de 2013 à http://pt.wikipedia.org/wiki/Simulated_annealing.

[8] Wikipedia. *Metropolis–Hastings algorithm.* Acesso em Junho de 2013 à http://en.wikipedia.org/wiki/Metropolis-Hastings_algorithm.

[9] Marcus Ritt, Luciana Buriol. *INF05010 — Otimização combinatória. Notas de aula.* 2012/04/04.